基本问题：

1.计算机图形学是什么

[计算机图形学](https://so.csdn.net/so/search?q=%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6&spm=1001.2101.3001.7020)是研究通过计算机将数据转换为图形，并在专门的显示设备上显示的原理，方法和技术的学科。

2.计算机图形学学什么

图形系统，画线圆，裁剪，填充，几何变换，几何造型，真实感图形

3.计算机图形学应用举例

①计算机辅助设计②科学可视化③计算机艺术④GIS⑤VR⑥计算机动画

4.计算机图形学主要内容

建模、渲染、动画和人机交互

5.图形的分类

几何图形，真实感图形

6.图形要素

几何要素，非几何要素

7.图形的表达方式

点阵法（图像），参数法（图形）

8.图形学应用生活中的例子

3D电影，3D游戏，医学影像，人脸识别，短视频特效

9.计算机图形学定义

计算机图形学是研究通过计算机将数据转换为图形，并在专门的显示设备上显示的原理，方法和技术的学科。

10.用户界面发展方向

高效化，简便化，多样化，现实化，人性化

11.影视中的变形技术要点

控制点，三角网

12.虚拟现实的一些技术

VR技术，AR增强现实，立体投影技术，全息雾幕投影

13.图形学发展关键词

图形显示器，交互式[图形学](https://so.csdn.net/so/search?q=%E5%9B%BE%E5%BD%A2%E5%AD%A6&spm=1001.2101.3001.7020)，光栅化，真实感，人机交互

14.专业知识和图形学相似的技术

几何校正，几何变换，DEM，TIN，遥感数字图像处理、三维分析与建模

15.图形学发展史关键词

①诞生：50年代，旋风一号

②确立：60年代，计算机图形学之父

③理论发展及标准化：70年代，光栅图形学，真实感图形学，实体造型技术，图形软件标准化

④实用化：80年代，图形应用软件

⑤广泛应用：90年代，几何造型，真实感图形绘制，人机交互，与计算机网络紧密结合

16.图形学之父

伊凡·萨瑟兰德

17.第一台显示器

旋风一号

18.第一台计算机

1946年的ENIAC

19.远程技术的发展成熟可能带来什么

远程教育的飞速发展等

20.图形学发展热点有哪些

造型技术，真实感图形绘制技术，人机交互技术，与计算机网络技术紧密结合

21.什么是LOD技术

对细节进行分层，提高效率

22.什么是IBR技术

以摄像机在现场拍摄的有限幅实景图像为样本，利用图像处理技术和视觉计算的方法，直接构建三维虚拟场景的技术

23.造型技术有哪些

规则形体：欧氏几何方法

不规则形体：分形几何，粒子系统，纹理映射，实体造型

24.图形学发展的历史有什么启示和思考

图形学的发展可以促进行业产能与相关关键技术的提高

25.列举几种VR动态感知设备

VR眼镜，VR头盔，VR衣服

26.等离子显示器原理

在显示屏上排列上千个密封的小低压气体室，通过电流激发使其发出肉眼看不见的紫外光，然后紫外光碰击后面玻璃上的红、绿、蓝3色荧光体发出肉眼能看到的可见光，以此成像。

27.液晶显示器原理

电光效应，当液晶收到电压影响时，就会改变它的物理性质而发生形变，此时通过它的光的折射角度就会发生变化，而产生色彩。

28.阴极射线管显示器原理

在真空中，利用电磁场产生高速的，经过聚焦的电子束，偏转到屏幕的不同位置，轰击屏幕表面的荧光材料而产生图形

29.彩色CRT的常用方法

射线穿透法，影孔板法

30.像素的概念

构成屏幕（图像）的最小元素

31.分辨率概念

一个阴极射线管在水平和垂直方向单位长度上能识别出的最大像素个数或显示器能显示像素的数量

32.什么是帧

一次扫描生成的影像，多少帧代表屏幕一秒刷新多少幅影像

33.什么是刷新率

指电子束对屏幕图像重复扫描的次数

34.喷墨打印机原理

通过数字电路控制打印喷头点阵喷射打印墨水完成打印的过程

35.激光打印机原理

利用激光扫描成像技术，使成像鼓充电带上静电后吸引碳粉，同时加热辊加热纸张把碳粉熔在纸张里面从而完成热成像的打印过程。

36.什么是液晶的电光效应

各向同性的透明物质在电场作用下显示出光学各向异性，物质的折射率因外加电场而发生变化的现象。

37.喷墨打印机喷墨方式分为哪些

压电式，气泡式，静电式，固体式

38.显示系统为哪两种

随机扫描系统，光栅扫描系统

39.什么是帧缓存器

计算机系统中专门用来存放正在合成或显示的图像的存储器

40.帧缓存容量怎么计算

分辨率M\*N，颜色个数K，显存大小V，单位为字节

41.视频控制器功能

控制显示设备，通过访问帧缓冲来刷新屏幕

42.图形加速卡组成

视频控制器，显存，显示处理器

43.图形扫描转换或光栅化的概念和步骤

确定一个象素集合及其颜色，用于显示一个图形的过程（从图元的参数表示形式转换成点阵表示形式的过程），称为图形的光栅化或扫描转换。

步骤：先确定有关象素，再用图形的颜色或其它属性，对象素进行某种写操作。

44.直线扫描转换常用的几种方法

DDA算法，中点画线法，Bresenham算法

45.DDA算法原理

即数值微分算法，是用数值方法解微分方程，通过同时对x和y各增加一个小增量，计算下一步的x，y的值

46.图元的生成概念，图元生成又叫什么

完成图元的参数表示形式到点阵表示形式的转换，又叫扫描转换图元

47.图形显示流程为哪两种

先裁剪再扫描转换；先扫描转换再裁剪

48.系统设计画线算法应考虑哪些方面

斜率，宽度，时间复杂度，空间复杂度

49.DDA函数程序代码

//直线DDA算法

void LineDDA(int x0, int y0, int x1, int y1, int color)

{

float dx, dy, y, k;

dy = y1 - y0;

dx = x1 - x0;

k = dy / dx;

y = y0;

for (int x = x0; x < x1; x++)

{

PutPixel(x, (int)(y + 0.5), color);

y += k;

}

}

50.中点画线算法的数学原理，及如何将数学原理转成算法

假定直线斜率k在0-1之间，在画直线段的过程中，当前像素点为(xp,yp)，下一个像素点有两种可选择点P1(xp+1,yp)或P2(xp+1,yp+1)。若M=(xp+1,yp+0.5)为P1与P2之中点，Q为理想直线与x=xp+1的交点。当M在Q的下方，则P2应为下一个像素点；M在Q的上方，应取P1为下一个像素点。

//中点画线法

void MidPointLine(int x0, int y0, int x1, int y1, int color)

{

int a, b, d1, d2, d, x, y;

a = y0 - y1;

b = y1 - y0;

d = a + a + b;

d1 = a + a; //取正右方像素

d2 = a + a + b + b; //取右上方像素

x = x0, y = y0;

PutPixel(x, y, color);

while (x < x1)

{

if (d < 0)

{

x++;

y++;

d += d2;

}

else

{

x++;

d += d1;

}

PutPixel(x, y, color);

}

}

51.Bresenham画线算法原理

过各行、各列像素中心构造一组虚拟网格线，按直线从起点到终点的顺序计算直线各垂直网格线的交点，然后确定该列像素中与此交点最近的像素

//Bresenham画线算法

void BresenhamLine(int x0, int y0, int x1, int y1, int color)

{

int x, y, dx, dy;

float k, e;

dx = x1 - x0;

dy = y1 - y0;

k = dy / dx;

e = -0.5;

x = x0, y = y0;

for (int i = 0; i <= dx; i++)

{

PutPixel(x, y, color);

x = x + 1;

e = e + k;

if(e>=0)

{

y++;

e = e - 1;

}

}

}

52.直线线宽处理

垂直线刷子，水平线刷子，方形刷子

53.圆生成原理

考虑中心在原点、半径为整数R的八分圆，圆的其他部分可以通过一系列的简单反射变换得到。

//中点画圆算法

void MidPointCircle(int x0, int y0, int r, int color)

{

int x = 0, y = r, d = 1.25 - r;

Circle8Points(x0, y0, x, y, color);

while (x<=y)

{

if (d < 0)

{

d += 2 \* x + 3;

x++;

}

else

{

d += 2 \* (x - y) + 5;

x++;

y--;

}

Circle8Points(x0, y0, x, y, color);

}

}

void Circle8Points(int x0, int y0, int x, int y, int color)

{

PutPixel(x0 + x, y0 + y, color);

PutPixel(x0 - x, y0 + y, color);

PutPixel(x0 + x, y0 - y, color);

PutPixel(x0 - x, y0 - y, color);

PutPixel(x0 + y, y0 + x, color);

PutPixel(x0 - y, y0 + x, color);

PutPixel(x0 + y, y0 - x, color);

PutPixel(x0 - y, y0 - x, color);

}

54.字符生成方法

点阵字符，向量字符

55.矩形填充算法，最核心的循环代码

//矩形填充算法

void FillRectangle(Rectangle\* rect, int color)

{

int x, y;

for (y = rect->ymin; y <= rect->ymax; y++)

{

for (x = rect->xmin; x <= rect->xmax; x++)

{

PutPixel(x, y, color);

}

}

}

56.多边形的表示方法有哪两种

顶点表示，点阵表示

57.描述逐点判断法中的射线法原理关键的要点

奇内偶外，异1同2

58.扫描线算法的关键点

求交，排序，配对，填色

59.活性边表中的四个元素有哪些

第1项，存当前扫描线与边的交点坐标x值;

第2项，存从当前扫描线到下一条扫描线间x的增量dx；

第3项，存该边所交的最高扫描线的y值ymax;

第4项，存指向下一条边的指针。

60.点阵表达的多边形又分为哪两种类型

内点表示，边界表示

61.写出种子填充算法的核心代码

//内点表示多边形的种子填充算法

void FloodFill4(int x, int y, int oldColor, int newColor)

{

if (GetPixel(x, y) == oldColor)

{

PutPixel(x, y, newColor);

FloodFill4(x, y + 1, oldColor, newColor);

FloodFill4(x, y - 1, oldColor, newColor);

FloodFill4(x - 1, y, oldColor, newColor);

FloodFill4(x + 1, y, oldColor, newColor);

}

}

//边界表示多边形的种子填充算法

void BoundaryFill4(int x, int y, int oldColor, int newColor)

{

int color;

color = GetPixel(x, y);

if ((color != oldColor) && (color != newColor))

{

PutPixel(x, y, newColor);

FloodFill4(x, y + 1, oldColor, newColor);

FloodFill4(x, y - 1, oldColor, newColor);

FloodFill4(x - 1, y, oldColor, newColor);

FloodFill4(x + 1, y, oldColor, newColor);

}

}

62.简单说明直线裁剪方法的基本原理

①线段是否全不在窗口内，若是，则弃，结束。

②线段是否全在窗口内，若是，则取，转到步骤4。

③计算该线段与窗口边界的交点，以此将线段分成两部分;丢弃不可见的部分，对剩下的部分转到步骤2。

④保留并显示该线段。

63.怎么理解CS直线裁剪方法的通过编码实现裁剪算法

将区域分成九个区域，按编码规则编码

**65.中点分割裁剪算法原理**

66.裁剪中的4位编码原理与运用

设左边区域第1位为1，右边三个第2位为1，下边三个第3位为1，上边三个第4位为1，窗口区域都为0

若两端点编码都为0，则在窗口内，无需裁剪；

若两端点编码求与为非0，则线段完全不可见；

若两端点编码求与为0，则可见性要进一步判断。

67.多边形SH裁剪算法原理

用窗口的每一条边去依次裁剪多边形，四条边裁剪完后的结果即为结果多边形

68.什么是图形变换，有什么用途

是指对图形的几何信息经过几何变换后产生新的图形；二维和三维图形的平移、旋转、比例和对称等几何变换，以及投影、透视变换等。

图形变换是计算机图形学的基础内容之一，通过图形变换，可由简单图形生成复杂图形，可用二维图形表示三维形体，甚至可对静态图化进行快速变换而数得图形的动态显示效果。

69.图形变换常见的类型

几何变换（二维三维），投影变换（平行投影，透视投影）；

二维变换矩阵，平移变换，比例变换，对称变换，旋转变换，错切变换，复合变换

70.几何变换的数学本质是什么

坐标系变换

71.什么是齐次坐标，怎么理解

齐次坐标就是将一个原本是n维的向量用一个n+1维向量来表示，是指一个用于投影几何里的坐标系统。

因为该坐标允许平移、旋转、缩放及透视投影等可表示为矩阵与向量相乘的一般向量运算。依据链式法则，任何此类运算的序列均可相乘为单一矩阵，从而实现简单且有效之处理。

齐次坐标表示是计算机图形学的重要手段之一，它既能够用来明确区分向量和点，同时也更易用于进行仿射（线性）几何变换。

72.写出二维，三维平移，旋转，缩放变换矩阵

二维平移：

1 0 0

0 1 0

tx ty 1

二维旋转：

Cos a sin a 0

-sin a cos a 0

0 0 1

二维缩放：

Sx 0 0

1. sy 0

0 0 1

三维平移：

1. 0 0 0
2. 1 0 0
3. 0 1 0

Tx tx tx 1

三维旋转：

绕X轴：

1 0 0 0

1. cos a sin a 0
2. -sin a cos a 0

0 0 0 1

绕Y轴：

Cos a 0 -sin a 0

0 1 0 0

Sin a 0 cos a 0

0 0 0 1

绕Z轴：

Cos a sin a 0 0

-sin a cos a 0 0

0 0 1 0

0 0 0 1

三维缩放：

Sx 0 0 0

0 sy 0 0

0 0 sz 0

(1-sx)xf (1-sy)xy (1-sz)xz 1

73.什么是复合变换，复合变换怎么实现

由两种及两种以上的基本变换组合而成的变换称为复合变换。

在连续变换时，先计算变换矩阵，再计算坐标

74.几何造型的基本概念

几何造型就是用计算机系统来表示、控制、分析和输出三维形体，表示形体的两种模型有用欧式几何描述的数据模型和用分形几何描述的过程模型。

75.几何造型的基本类型

数据模型和过程模型。

数据模型分为线框模型、表面模型、实体模型。

过程模型包括：分形几何，随机插值模型、迭代函数系统、L系统、粒子系统等。

76.常见的几何造型方法有哪些

线框造型系统，曲面造型系统，实体造型系统

77.几何变换中，任意参照点的旋转实现步骤

①平移对象使参照(基准)点移到原点

②绕坐标原点旋转

③平移对象使基准点回到原始位置

78.列举几种常见的形体表示模型

①扫描变换(Sweep)表示模型

②构造实体几何(CSG)表示模型

③边界表示(B-Rep)模型

④空间分割模型

79.列举几种常见过程模型，理解原理思想

①分形几何

描述非规则的自然事物

②随机插值模型

不是事先决定各种图素和尺度，而是用一个随机过程的采样路径作为构造模型的手段。

③迭代函数系统

生成规则：取一等边三角形，连接各边中点将原三角形分成四个小三角形，然后舍弃位于中间的一个小三角形，将剩下的其余三个小三角形按同样方法继续分割，并舍弃位于中间的那个三角形，如此不断地分割与舍弃，就能得到中间有大量孔隙的Sierpinski垫片。

④基于文法的模型：L系统

用文法表示植物的拓扑结构；通过图形学方法生成逼真的画面。

⑤粒子系统

用大量的粒子图元(Particle)来描述景物。每个粒子都有一组随机取值的属性，如起始位置、初速度、颜色及大小。

80.真实感图形绘制的主要步骤

①虚拟场景标识

②取景变换和光栅化

③消隐

④光照计算

81.解释消隐技术的Z-buffer算法原理

①先将Z缓冲器中个单元的初始值置为最小值。

②当要改变某个像素的颜色值时，首先检查当前多边形的深度值是否大于该像素原来的深度值（保存在该像素所对应的Z缓冲器的单元中）

A：如果大于，说明当前多边形更靠近观察点，用它的颜色替换像素原来的颜色；

B：否则说明在当前像素处，当前多边形被前面所绘制的多边形遮挡了，是不可见的，像素的颜色值不改变

82.理解并解释光照模型

根据光学物理的有关定律，计算物体表面各点投射到观察者眼中的光线的光亮度和色彩组成的数学公式。

光照模型就是一个数学模型，我们可以用这个数学模型求出物体表面每一点的亮度，因此它也叫浓淡模型。

83.着色方法类型和优缺点比较

①均匀着色

任取多边形上一点，利用光照明方程计算出它的颜色，用这个颜色填充整个多边形

优点：每个多边形只需计算一次光照明方程，速度快

缺点：相邻多边形颜色过渡不光滑

②光滑着色

采用插值方法，主要有Gouraud(高洛德）方法和Phone(冯)方法

优点：颜色过渡均匀，图形较平滑

缺点：计算量大

84.理解并解释纹理映射技术

纹理映射是将纹理图象值映射到三维物体表面的技术，实质上是建立纹理和三维物体之间的对应关系式。

第一章-导论

1. [计算机图形学](https://so.csdn.net/so/search?q=%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%9B%BE%E5%BD%A2%E5%AD%A6&spm=1001.2101.3001.7020)的定义

计算机图形学是一门研究怎样用计算机表示、生成、处理和显示图形的学科。图形主要分为两类：基于线框模型描述的几何图形，基于表面模型描述的真实感图形。

图形的表示方法：

1. 参数法，在设计阶段建立几何模型时，用形状参数和属性参数描述图形的方法，形状参数可以是直线段的起点，终点等几何参数，属性参数则包括直线段的颜色，线形，宽度等非几何参数。例如y=kx+b
2. 点阵法，在绘制阶段用具有颜色信息的像素点阵来表示图形的一种方法，所描述的图形成为图像。

计算机图形学就是研究将图形的表示法从参数法换到点阵法的一门学科。

2. 计算机图形学的应用领域

1. 计算机游戏
2. 计算机辅助设计（CAD）
3. 计算机艺术（动画，图像渐变）
4. 虚拟现实（VR）
5. 计算机辅助教学
6. 三维或高维数据的可视化

2.1 计算机图形学与其他学科的关系

相关学科：图像处理，模式识别。

* 计算机图形学：研究如何用计算机把描述图形的几何模型通过指定算法转化为图像显示的一门学科。例如3ds max
* 图像处理：对数字图像进行增强，去噪，复原，分割，重建，编码，存储，压缩等等，例如Photoshop
* 模式识别：对点阵图像进行特征抽取，利用统计学方法给出图形描述的学科，例如输入法手写打字

3. 图形显示器的发展及其工作原理

图形显示器是计算机图形学发展的硬件依托，发展经历了随即扫描显示器，直视存储管显示器，目前广泛应用的光栅扫描显示器。

3.1 阴极射线管（[CRT](https://so.csdn.net/so/search?q=CRT&spm=1001.2101.3001.7020" \t "_blank)）

* CRT是光栅扫描显示器的显示部件，其功能与电视机的显像管类似，主要是由电子枪、偏转系统、荫罩、荧光粉层及玻璃外壳五部分组成
* 电子枪是由灯丝、阴极、控制栅组成。彩色CRT中有红绿蓝三支电子枪。有的显示器的电子枪是单枪三束
* CRT通电后灯丝发热，阴极被激发射出电子，电子受到控制栅的调节形成电子束。电子束经聚焦系统聚焦后以高速轰击到荧光屏上，荧光粉层被激发后发出辉光形成一个光点。
* CRT偏转系统可以控制电子束在指定的位置上轰击荧光屏，整个荧光屏依次扫描完毕后，图像显示完成。
* 由于荧光粉具有余辉特性——电子束停止轰击荧光屏后，荧光粉的亮度并不是立即消失，而是按指数规律衰减，图像逐渐变暗，为了得到亮度稳定的图像，电子枪需要不断地根据帧缓存的内容轰击荧光屏，反复重绘同一幅图像，即不断刷新屏幕。

3.2 随机扫描显示器

* 20世纪60年代中期出现随机扫描显示器。电子束只在屏幕上显示图形的部分移动。
* 图像的定义是存放在文件存储器中的一组画线命令。随机扫描显示器周期性地读取画线命令，依次在屏幕上画出线条，当所有的画线命令都执行完毕后，图像就显示出来。这时随机扫描显示器又返回到第一条命令行进行屏幕刷新。

3.3 直视存储管显示器

* 70年代后期发展了利用CRT本身来存储信息，而且不再需要刷新屏幕的显示器。
* DVST使用紧贴在荧光层后的存储栅的电荷分布来存储图形。DVST使用两支电子枪，一支是写电子枪，用来存储图形；另一支是读电子枪，用来图形显示。
* 从表面上看DVST极象是一个长余辉的CRT，一条线一旦画在屏幕上，在一小时之内都将是可见的。

缺点：

* 不能显示彩色，而且不能局部修改图像。要擦除图像的某一部分，必须先擦除整个屏幕，然后重画修改后的图像，从而妨碍了动态图形的生成。
* 因为不是连续地刷新屏幕，就不能用光笔进行交互操作；
* 屏幕的反差较弱，一小时后图像就看不清楚了

3.4 光栅扫描显示器

画点设备：

* 光栅扫描显示器电子束的强度可以不断变化，所以容易生成颜色连续变化的真实感图像。光栅扫描显示器是画点设备，可看作是一个点阵单元发生器，并可控制每个点阵单元的亮度，这些点阵单元被称为像素（Pixel）。
* 光栅扫描显示器不能从单元阵列中的一个可编址的像素点直接画一段直线到达另一个可编址的像素点，只能用靠近这段直线路径的像素点集来近似地表示这段直线。

扫描线：

光栅显示器为了在能整个屏幕上显示出图形，电子束需要从屏幕的左上角开始，沿着水平方向从左至右匀速地扫描，一直扫描到屏幕的右下角，显示出一帧图像。

位面与帧缓冲：

光栅扫描显示器总是与帧缓冲存储器（frame buffer，简称为帧缓冲）联系在一起。帧缓冲中单元数目与显示器上的像素数目相同，而且单元与像素一一对应。帧缓冲中所存储的信息就是屏幕上对应位置的图像信息。要在屏幕上显示图像，首先要把图像信息写入帧缓冲，然后光栅扫描显示器访问帧缓冲，再把其中的内容显示到屏幕上。帧缓冲单元的数值决定了显示器像素的颜色。显示器颜色的种类与帧缓冲中每个单元的位数有关。

* 如果屏幕上每个像素的颜色只用一位（Bit）表示，其值非0即1，屏幕只能显示黑白二色图像，称为黑白显示器，此时帧缓冲器只有一个位面。
* 如果每个像素的颜色可以用一个字节（Byte）8bit表示，帧缓冲器需要用八个位面，可表示256种灰度，称为灰度显示器。
* 如果每个像素用R、G、B三原色混合表示，其中每种原色分别用一个字节表示，各对应一把电子枪，每种颜色可有256种亮度，三种颜色的组合是2^24颜色，共有24个位面。实际使用的是32个位面（32bit），也就是说用4个字节表示颜色，最后一个字节用于表示颜色融合，称为RGBA模型。

索引色：

为了进一步提高颜色的种类，控制帧缓冲的增加，可把帧缓冲中的位面号作为颜色索引表的索引号，为每组原色配置一个颜色索引表，颜色索引表有256项，每一项具有w位字宽，当w大于8时，如w=10，可以有210种亮度等级，但每次只能有256种不同的亮度等级可用，这种颜色称为索引色。

4. 图形软件标准的形成

图形软件标准最初是为提高软件的可移植性而提出的。早期各硬件厂商基于自己生产的图形显示设备开发的图形软件包是为其专用设备提供的，彼此互不兼容，不能直接移植到另一个硬件系统上使用。

1974年，ISO批准的第一个图形软件标准软件：图形核心系统（Graphics Kernal System，GKS），它是二维图形软件标准。1988年，ISO批准的GKS3D是三维图形软件标准。

5. 三维图形[渲染管线](https://so.csdn.net/so/search?q=%E6%B8%B2%E6%9F%93%E7%AE%A1%E7%BA%BF&spm=1001.2101.3001.7020" \t "_blank)

* 将三维场景转化为一幅二维图像的过程。
* 图像中物体所处位置及外形由其几何数据和摄像机的位置共同决定，物体外表是受到其材质属性、光源、纹理及着色模型所影响。
* 渲染管线的基本结构包括3个阶段：应用程序、几何、光栅化。
* 每个阶段本身也可能是一条管线，还可以对有的阶段进行全部或者部分的并行化处理，如光栅化阶段。

计算机图形学概述

> 计算机图形学主要研究内容就是研究如何在计算机中表示图形、以及利用计算机进行图形计算、处理和显示的相关原理和算法。图形通常由点、线、面、体等几何元素和灰度、色彩、线型、线宽等非几何属性组成。从处理技术上来看，图形主要分两类，一类是基于线条信息表示的，如工程图、等高线地图、曲面和线框图等，另一类是明暗图，也就是通常所说的真实感图形。

> 计算机图形学一个主要的目的就是要利用计算机产生令人赏心悦目的真实感图形。为此，必须创建图形所描述的场景的几何表示，再用某种光照模型，计算在假想的光源、纹理、材质属性下的光照明效果。所以计算机图形学与另一门学科计算机辅助几何设计有着密切的关系。事实上，图形学也把可以表示几何场景的曲线曲面造型技术和实体造型技术作为其主要的研究内容。

> 同时，真实感图形计算的结果是以数字图象的方式提供的，计算机图形学也就和图像处理有着密切的关系。图形与图像两个概念间的区别越来越模糊，但还是有区别的：图像纯指计算机内以位图形式存在的灰度信息，而图形含有几何属性，或者说更强调场景的几何表示，是由场景的几何模型和景物的物理属性共同组成的。 \*——摘自维基百科\*

图形学领域

\* \*\*建模\*\*：对形状和外观性质进行数学定义，并且能够存储在计算机中。

\* \*\*渲染\*\*：根据三维计算机模型生成带阴影的图像。

\* \*\*动画\*\*：利用图像序列，产生物体运动的视觉效果的一种技术。这里也要用到建模和绘制，但控制时间这个关键问题在基本建模和绘制中一般是不考虑的。

\* \*\*人机交互\*\*：在输入设备（鼠标、写字板）、应用程序和以图形或者其他感官方式向用户发送的反馈之间建立接口。历史上，该领域和图形学关系非常密切。

\* \*\*可视化\*\*：视图通过视觉显示让用户看得更明白。可视化问题一般都会涉及图形学。

\* \*\*图像处理\*\*：对二维图形进行处理，图形学和视觉领域都要用到。

\* \*\*三维扫描\*\*：利用测距技术建立三维测量模型。这种模型在创建细节丰富的视觉图像方面很有用处，二模型处理常常用到图形学算法。

表示图形的方法

\* \*\*点阵表示\*\*：简称为图像（数字图像）—— 枚举出图形中所有的点。

\* \*\*参数表示\*\*：简称为图形——由图形的参数（方程或分析表达式的系数，线段的端点坐标等）+ 属性参数（颜色、线型等）来表示图形。

图形显示设备

\* 阴极射线管（CRT）

\* 光栅扫描显示系统 —— 影孔板法

\* 随机扫描显示系统

\* 彩色CRT监视器

\* 平板显示器

>阴极射线管（CRT）显示原理

> 工作原理：高速的电子束由电子枪发出，经过聚焦系统、加速系统和磁偏转系统就会到达荧光屏的特定位置。电子束轰击到荧光屏的不同部位，被其内表面的荧光物质吸收，发光产生可见的图形。荧光物质受电子束一次轰击之后发光，亮度会迅速衰减，所以要保持显示一副稳定的画面，必须不断地发射电子束（不断刷新）

> \*\*相关概念\*\*

> 1. 刷新频率：每秒钟重绘屏幕的次数。刷新一次是指电子束从上到下扫描一次的过程。

> 2. 像素：构成屏幕（图像）的最小元素

> 3. 分辨率：CRT在水平或竖直方向单位长度上能识别的最大像素个数，即水平和垂直方向上每厘米可绘制的点数（单位dpi）。

> \*\*彩色阴极射线管显示原理\*\*

> 原理：通过将能发不同颜色的光的荧光物质进行组合而产生彩色的。彩色CRT显示器的荧光屏上涂有三种荧光物质，它们分别能发红、绿、蓝三种颜色的光。而电子枪也发出三束电子束激发这三种物质，中间通过一个控制栅格来决定三束电子到达的位置。三束电子经过影控板的选择，分别到达三个荧光点的位置。通过控制三个电子束的强弱就能控制屏幕上点的颜色。

> 每个电子枪发出的电子束的强度有256个等级（灰度），则显示器能同时显示256\*256\*256=16M种颜色，称为真彩色。

> \*\*光栅图形显示系统\*\*

> 1. 显示处理器：其主要任务是将应用程序定义为一组像素强度值，存放在帧缓冲存储器（扫描转换），同时也能执行某些附加的操作几何变化、裁剪、纹理映射等等。

> 2. 帧缓冲存储器：简称帧缓冲器，俗称显存，保存对应屏幕所有点的亮度值。

> 3. 视频控制器：建立帧缓存与屏幕像素之间的一一对应，负责刷新。

第二章-基本图元的扫描转换

1. 光栅化

光栅图形显示器可以看作一个像素的矩阵。在光栅显示器上显示任何一种图形，实际上都是一些具有一种或多种颜色的像素集合。

确定最佳逼近图形的像素集合，并用指定属性写像素的过程称为图形扫描转换或光栅化。

2. 点的扫描转换

将应用程序中得到的点的坐标，映射到屏幕上的像素坐标。采用整数值表示像素的坐标。

3. 直线的扫描转换

* 直线的扫描转换就是在屏幕像素点阵中确定最佳逼近理想直线的像素点集的过程。计算机图形学要求直线的绘制速度要快，尽量使用加减法，避免乘除，开方，三角的等复杂运算。
* 数学：理想的直线是没有宽度和端点的。
* 图形学：直线是有两个端点，有宽度的。

3.1 直线基础

1. 直线的斜截式方程：  
   y=kx+b
2. 直线的两点式方程：

(y-y1)/(x-x1)=(y1-y2)/(x1-x2)

3.2 数值微分分析法（DDA）

过点（x1,y1）,(x2,y2)的直线段y = kx + b

递推公式（由x方向增量引起y的改变）：

x i+1=xi + Δx

y i+1=yi+Δx(y2-y1)/(x2-x1)

算法思想

选定x2－x1和y2－y1中较大者作为步进方向(假设x2－x1较大)，取该方向上的增量为一个象素单位(△x=1)，然后利用式(2－1)计算另一个方向的增量(△y=△x·K=K)。通过递推公式把每次计算出的(x\_i+1,y\_i+1)取整后送到显示器输出，则得到扫描转换后的直线。

　　之所以取x2－x1和y2－y1中较大者作为步进方向，是考虑沿着线段分布的象素应均匀。

　　另外，算法实现中还应注意直线的生成方向，以决定Δx及Δy是取正值还是负值。

算法特点

该算法简单，实现容易，但由于在循环中涉及实型数的运算，因此生成直线的速度较慢。

算法实现

void CDDALineDlg::DDA\_Line(int x1, int y1, int x2, int y2, int color)

{

CPaintDC dc(this); //获得窗口句柄

float increX,increY; //x，y方向的增量

float x,y;

int steps;   //循环次数，即画的点数

steps = max(abs(x2-x1),abs(y2-y1));//选较大者作为步进方向

increX = (float)(x2-x1)/steps;//x方向增量

increY=(float)(y2-y1)/steps;//y方向增量

x=x1;

y=y1;

for(int i=1;i<=steps;i++){

SetPixel(dc,x,y,color); //在(x，y)处，以color色画点

x+=increX;

y+=increY;

}

}

3.3 Bresenham直线转换法

3.3.1算法原理

每次在主位移方向上走一步，另一个方向上走不走取决于中点误差项的值。

给定理想直线的起点坐标为 (x0, y0) , 终点坐标为 (x1, y1)，则直线的隐函数方程为：  
F(x,y)=y-kx-b=0

3.3.2 整数化处理

上述算法缺点：在计算误差项 d 时，其初始值何递推公式包含浮点数0.5和斜率 k 。可以将递推公式放大2Δx倍，变成整数运算。

0 \leq k \leq 1 ，x 为主步进方向

误差项初始值：d_0 = \Delta x-2\Delta y

当 di < 0 ：d_{i+1}=d_i+2\Delta-2\Delta y

当 di >= 0 ：d_{i+1}=d_i-2\Delta y

当k > 1，y 为主步进方向

误差项初始值：d_0 = \Delta y-2\Delta x

当 di < 0 ： d_{i+1}=d_i+2\Delta y-2\Delta x

当 di >= 0 ：d_{i+1}=d_i-2\Delta x

3.4 圆的扫描转换

圆的扫描转换就是在屏幕像素点阵中确定最佳逼近于理想圆的像素点集的过程。圆的绘制可以使用简单方程画圆算法或极坐标画圆算法，但涉及开方或三角运算效率很低。本节主要讲解1/8圆的中点Bresenham算法原理。

算法原理

圆心在原点，半径为 R 的隐函数表达式：F(x,y)=x^2 +y^2-R^2=0

圆将平面划分为3个区域：

* 圆上的点 F(x,y)=0
* 圆外的点 F(x,y)>0
* 圆外的点 F(x,y)<0

递推公式

1. 中点误差项初始值 d_0=F(1,R-0.5)=1.25-R
2. 当di < 0 d_{i+1}=d_i+2x_i+3
3. 当di >= 0 d_{i+1}=d_i+2(x_i-y_i)+5

4. Wu反走样算法

4.1 算法原理

Wu反走样算法是根据像素与理想直线的距离对相邻两个像素的亮度等级进行调节。算法采

用空间混色原理对走样现象进行修正。空间混色原理指出，人眼对于某一区域颜色的识别是取这个区域颜色的平均值。

直线的绘制算法

\*\*直线的绘制算法\*\*

\* 直线的扫描转换

\* DDA 算法

\* Bresenham 算法

\* 中点坐标算法

1. 直线的扫描转换

\*\*数学直线\*\*：在数学上，理想的直线是一条没有宽度，由无穷多个无限小的连续的点组成。

\*\*光栅平面显示的图像\*\*：在光栅显示平面上，我们只能用二维光栅网格上尽可能靠近这条直线的象素集合来表示它。每个象素具有一定的尺寸，是显示平面上可被访问的最小单位，它的坐标x和y只能是整数，也就是说相邻象素的坐标值是阶跃的而不是连续的。

直线的扫描转换算法就是利用直线的数学性质，通过直线方程，算出直线在栅格上每个点的坐标。

\*\*缺点\*\*：算法中涉及大量的浮点运算，效率不高；没有利用栅格显示的特性。

2. DDA 算法

DDA（digital differential analyzer ） 算法，采用了增量的方式，提高直线扫描转换的效率。在计算 Y<sub>x+1</sub> 时，只需要用到 Y<sub>x</sub> 的值。

2.1 基本思想

假设直线段的宽度为1,直线段的斜率: |k|<=1

已知过端点的直线段L：y=kx+B

直线斜率为：

> $$k=(y\_1-y\_0)/ (x\_1-x\_0)$$

计算：

> $$

Y\_{i+1} = kX\_{i+1} + B

        = k(X\_i+D\_x)+B

        = kX\_i + kD\_x +B

        = Y\_i + kD\_x

$$

在栅格上，X 的增量 D<sub>x</sub> 的值都是1，所以

> $$ Y\_{i+1} = Y\_i + k $$

即：当x每递增1，y递增k，(即直线斜率)；取象素点(x, round(y))作为当前点的坐标。

2.2 算法实现

DDALine(int x0, int y0, int x1, int y1, int color)

{

    int x

    float dx,dy,k,y

    dx=x1-x0; dy=y1-y0;

    k=dy/dx;

    y=y0;

    for(x=x0;x<=x1;x++)

    {

        drawpixel(x,int(y+0.5),color);

        y=y+k;

    }

}

注：这里的 DDA 算法只实现了 |k|<=1 的情况，而 |k|>1 的直线只需要利用对称变换得到。

DDA 算法简单，实现容易，但y与k必须用浮点数表示，而且每一步都要对y进行四舍五入后取整，这使得它不利于硬件实现。

3. Bresenham 算法

3.1 基本思想

过各行各列象素中心构造一组虚拟网格线。按直线从起点到终点的顺序计算直线与各垂直网格线的交点，然后根据误差项的符号确定该列象素中与此交点最近的象素。该算法的巧妙之处在于采用增量计算，使得对于每一列，只要检查一个误差项的符号，就可以确定该列的所求象素。

先考虑斜率k=dy/dx≤1的直线。设直线方程为：

> $$ y\_{i+1}=y\_i+k(x\_{x+1}-x\_i)=y\_i+k, k=dy/dx $$

假设当前像素的 x 坐标已经确定为 x<sub>i</sub> ，其 y 坐标为 y<sub>i</sub>，由于坐标 (x<sub>i</sub>,y<sub>i</sub>) (i=0,1,2,....) 只能取整数，那么下一个像素的 x 坐标为：

> $$ x\_{i+1} = x\_i + 1 $$

而 y<sub>i+1</sub> 的坐标有两种可能：

1. 保持不变，即 y<sub>i+1</sub> = y<sub>i</sub>

2. y 坐标递增1，即 y<sub>i+1</sub> = y<sub>i</sub> + 1

设 A 为 CD 边的中点，正确的选择：若 B 点在 A 点上方，选择 D 点；否则，选 C 点。

因为直线的起始点在象素中心，所以误差项d的初值d0＝0。X下标每增加1，d的值相应递增直线的斜率值k，即d＝d＋k。一旦d≥1，就把它减去1，这样保证d在0、1之间。

\* 当 d≥0.5 时，最接近于当前象素的右上方象素 (x<sub>i</sub>+1, y<sub>i</sub>+1)

\* 而当 d<0.5 时，更接近于右下方象素 (x<sub>i</sub>+1, y<sub>i</sub>)

为了方便计算，令 e=d-0.5，e的初值为-0.5，增量为k。

\* 当 e≥0 时，取右上方象素 (x<sub>i</sub>+1, y<sub>i</sub>+1)

\* 而当d<0.5时，取右方象素 (x<sub>i</sub>+1, y<sub>i</sub>

3.2 算法实现

void Bresenhamline (int x0,int y0,int x1, int y1,int color)

{

    int x, y, dx, dy;

    float k, e;

    dx = x1-x0, dy = y1- y0, k=dy/dx;

    e=-0.5, x=x0, y=y0;

    for (i=0; i<=dx; i++)

    {

        drawpixel (x, y, color);

        x=x+1，e=e+k;

        if (e>=0)

        { y++, e=e-1;}

    }

}

注：这里的 Bresenham 算法只实现了 0< k <=1 的情况，而其他斜率的直线只需要利用对称变换得到。

圆和椭圆的绘制算法

\*\*圆的绘制算法\*\*

\* 圆的扫描转换

\* 中点画圆算法

\* Bresenham 算法

\*\*椭圆的绘制算法\*\*

\* 中点坐标算法

1. 圆的扫描转换

与直线的扫描转换一样，通过圆的方程绘制圆，这样因为涉及到开方等运算，效率很低，不便于硬件实现，所以需要设计更加高效的算法。

2. 中点画圆算法

2.1 八对成称画圆

在介绍画圆算法前，介绍一下八对称画圆，这不是什么高深的算法，只是利用了圆的对称性，在实际画圆时只需要计算 1/8 圆就可以了。

void CirclePoints(int x, int y, Color color)

{

    drawPixel(x,y,color);

    drawPixel(x,-y,color);

    drawPixel(-x,y,color);

    drawPixel(-x,-y,color);

    drawPixel(y,x,color);

    drawPixel(-y,x,color);

    drawPixel(y,-x,color);

    drawPixel(-y,-x,color);

}

2.2 算法思想

与直线的中点算法一样，圆的中点算法也是一个增量算法。

我们假设圆的圆心在原点，半径为 R ，我们绘制的第一个像素时圆的最顶部的那个点 (0,R)。我们绘制下一个点的依据是栅格的中点 M 是在圆内还是在圆外，即我们现在要画 (x<sub>p+1</sub>, y<sub>p+1</sub>) 这个点，那么就判断点 (x<sub>p</sub> + 1, y<sub>p</sub> - 0.5) 是在圆内还是在圆外。

\* 如果在圆内就绘制( x<sub>p</sub> + 1, y<sub>p</sub> )

\* 如果在圆外就绘制( x<sub>p</sub> + 1, y<sub>p</sub> + 1 )

我们构造判别式：

> $$ f\_{cir}(x,y) = x^2 + y^2 - r^2 $$

将点 M (x<sub>p</sub> + 1, y<sub>p</sub> - 0.5) 带入判别式，得到：

> $$ d = f\_{cir}(x\_p+1,y\_p-0.5) = (x\_p+1)^2 + (y\_p-0.5)^2 - R^2 $$

\* 当d<0时，M在圆内，这说明点P1距离圆更近， 应取点P1作为下一象素点；

\* 当d>0时，M在圆外，表明P2点离圆更近，应取P2点；

\* 当d=0时，在P1点与P2点之中随便取一个即可，我们约定取P2点。

若 d<0,  则取P1为下一象素，而且再下一象素的判别式为：

> $$ d'= f\_{cir}(x\_p+2, y\_p-0.5) = (x\_p+2)^2 + (y\_p-0.5)^2 - R^2 = d + 2x\_p + 3 $$

若d>=0, 则应取P2为下一象素，而且下一象素的判别式为：

> $$ d'= f\_{cir}(x\_p+2, y\_p-1.5) = (x\_p+2)^2 + (y\_p-1.5)^2 - R^2 = d + 2(x\_p-y\_p) + 5 $$

由于第一个像素是 (0,R) ， 判别式 d 的初始值：

> $$ d\_0 = f\_{cir}(1, R-0.5) = 1.25 - R$$

2.3 算法实现

MidPointCircle(int r int color)

{

    int x,y;

    float d;

    x=0; y=r; d=1.25-r;

    circlepoints (x,y,color); //显示圆弧上的八个对称点

    while(x<=y)

    {

        if(d<0)

            d+=2\*x+3;

        else

        {

            d+=2\*(x-y)+5;

            y--;

        }

        x++;

        circlepoints (x,y,color);

    }

}

注：为了进一步提高算法的效率，可以将上面的算法中的浮点数改写成整数，将乘法运算改成加法运算，即仅用整数实现中点画圆法。初始化运算使用 e=d-0.25 代替 d ，对应于 e=1-R。

3. Bresenham 算法

3.1 基本思想

以点 (0,R) 为起点按顺时针方向生成圆，则在第一象限 y 是 x 的单调递减函数。假设圆心和起点均精确地落在像素中心上。如果已经知道圆弧上的一点 (x<sub>i</sub>, y<sub>i</sub>)，下一个像素的选取有三种可能：

1. 正右方像素H

2. 右下角像素D

3. 正下方像素V

构造函数：

> $$ f(x,y) = x^2 + y^2 - R^2 $$

这三个像素的偏差的平方为：

> $$ \Delta\_H = (x+1)^2 + y^2 - R^2 $$

> $$ \Delta\_D = (x+1)^2 + (y-1)^2 - R^2 $$

> $$ \Delta\_V = x^2 + (y-1)^2 - R^2 $$

令

> $$ \sigma\_{HD} = |\Delta\_H| - |\Delta\_D| = 2(\Delta\_D+y\_i) - 1$$

> $$ \sigma\_{DV} = |\Delta\_D| - |\Delta\_V| = 2(\Delta\_D-x\_i) - 1 $$

\* 当 Delta\_D < 0 时，如果 HD < 0 ，则取 H 下一个像素点，否则取V为下一个像素点。

\* 当 Delta\_D > 0 时，如果 DV < 0 ，则取 D 下一个像素点，否则取V为下一个像素点。

\* 当 Delta\_D = 0 时，则取 D 下一个像素点。

\*\*推导递推公式：\*\*

1. 如果取正右方像素（H）为下一个像素点，则第(i+1)个像素的坐标和有关的偏差的递推公式为：

> $$ x\_{i+1} = x\_i + 1 $$

> $$ y\_{i+1} = y\_i $$

> $$ \Delta D\_{i+1} = \Delta D\_{i} + 2x\_{i+1} + 1 $$

2. 如果取右下角的像素（D）为下一个像素点，则第(i+1)个像素的坐标和有关的偏差的递推公式为：

> $$ x\_{i+1} = x\_i + 1 $$

> $$ y\_{i+1} = y\_i - 1 $$

> $$ \Delta D\_{i+1} = \Delta D\_{i} + 2x\_{i+1} - 2y\_{i+1} + 2 $$

3. 如果取正下方的像素（V）为下一个像素点，则第(i+1)个像素的坐标和有关的偏差的递推公式为：

> $$ x\_{i+1} = x\_i $$

> $$ y\_{i+1} = y\_i - 1 $$

> $$ \Delta D\_{i+1} = \Delta D\_{i} + 2y\_{i+1} + 1 $$

3.2 算法实现

Bresenham\_Circle(int r, Color color)

{

    int x,y,delta,delta 1,delta 2,direction;

    x=0; y=r;

    delta = 2\*(1-r);  //△d的初始值

    Limit =0;

    while (y>=Limit)

    {

        drwpixel(x,y,color)

        if (delta <0)

        {

            delta 1= 2\*(delta +y)-1;  // 取H点

            if (delta 1 <=0 )direction =1;

            else direction=2; //取D点

        }

        else if (delta>0)

        {

               delta 2 = 2\*(delta –x)-1;  // 取D点

               if(delta 2 <0) direction =2;

               else direction =3; //取V点

        }

        else

          direction =2;

        Switch (direction)

        {

            case 1:

                x++;

                delta +=2\*x+1;

                break;

            case 2:

                x++;

                y--;

                delta +=2\*(x-y+1);

                break;

            case 3:

                y--;

                delta +=(-2\*y+1);

                break;

        }

    }

}

4. 椭圆的中点画法

4.1 基本思想

> $$ f(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0 $$

椭圆的对称性，只考虑第一象限椭圆弧生成，分上下两部分，以切线斜率为-1的点作为分界点。

\* 在上部分（区域2），法向量的y向分量较大，斜率K满足|k|<1, |△x|＞|△y|，所以x方向为主位移方向

\* 在下部分（区域1），法向量的x分量较大，斜率K满足|k|>1, |△y|＞|△x|，所以y方向为主位移方向。

与圆弧中点算法类似：确定一个像素后，接着在两个候选像素的中点计算一个判别式的值，由判别式的符号确定更近的点。

\*\*先讨论椭圆弧的上半部分\*\*，(Xp, Yp) 中点 (Xp+1, Yp-0.5)

> $$ d\_1 = F(X\_p+1, Y\_p-0.5) = b^2(X\_p+1)^2 + a^2(Y\_p-0.5)^2 - a^2b^2 $$

根据 d1 的符号来决定下一个像素是取正右方的那个，还是右上方的那个

\* 若 d1<0 ，中点在椭圆内，取正右方象素，判别式更新为：

> $$ d1'=F(X\_p+2,Y\_p-0.5)=d1+b^2(2X\_p+3) $$

d1的增量为

> $$ b^2(2X\_p+3) $$

\* 当d1≥0，中点在椭圆外，取右下方象素，更新判别式：

> $$ d1'=F(X\_p+2,Y\_p-1.5)=d1+b^2(2X\_p+3)+a^2(-2Y\_p+2) $$

d1的增量为

> $$ b^2(2X\_p+3)+a^2(-2Y\_p+2) $$

判别式的初始值：

> $$ d\_0=F(1,b-0.5)=b^2+a^2(b-0.5)^2-a^2b^2=b^2+a^2(-b+0.25) $$

\*\*从椭圆弧的上半部分转入转入下半部分判断条件\*\*：

\* 上半部分椭圆弧一点P(xi,yi)，中点M(xi+1,yi-0.5)，满足x方向分量小于y方向分量：

> $$ b^2(x\_i+1)<a^2(y\_i-0.5) $$

\* 而在下一个中点，不等号改变方向，说明椭圆弧从上半部分转入到下半部分

> $$ b^2(x\_i+1)>a^2(y\_i-0.5) $$

\*\*再来推导椭圆弧下半部分的绘制公式\*\*：

> $$d\_2 = F(x\_p+0.5, y\_p-1) = b^2(x\_p + 0.5)^2 + a^2(y\_p-1)^2 - a^2b^2$$

\* 若 d<sub>2</sub> > 0，取 P1(X<sub>p</sub>,Y<sub>p</sub>-1)，递推公式：

> $$ d\_2'=F(x\_p + 0.5, y\_p - 2) = b^2(x\_p+0.5)^2 + a^2(y\_p-2)^2 - a^2b^3 = d\_2 + a^2(-2y\_p+3) $$

\* 若 d<sub>2</sub> <= 0，取 P1(X<sub>p</sub> + 1,Y<sub>p</sub>-1)，递推公式：

> $$ d\_2'=F(x\_p + 1.5, y\_p - 2) = b^2(x\_p+1.5)^2 + a^2(y\_p-2)^2 - a^2b^3 = d\_2 + b^2(2x\_p+2) + a^2(-2y\_p+3) $$

用上半部分计算的最后点 (x,y) 来计算下半部分中 d 的初始值：

> $$ d=b^2(x+0.5)^2 + a^2(y-1)^2 - a^2b^2 $$

4.2 算法实现

void MiddlePointOval(x0,y0,a,b,color){

    var x=a;

    var y=0;

    var taa = a\*a;

    var t2aa = 2 \* taa;

    var t4aa = 2 \* t2aa;

    var tbb = b\*b;

    var t2bb = 2\*tbb;

    var t4bb = 2\*t2bb;

    var t2abb = a \* t2bb;

    var t2bbx = t2bb \* x;

    var tx = x;

    var d1 = t2bbx \* (x-1) + tbb / 2 + t2aa \* (1-tbb);

    while(t2bb \* tx > t2aa \* y){

        CirclePoint4(x,y,x0,y0,color);

        if(d1<0){

            y += 1;

            d1 = d1 + t4aa \* y + t2aa;

            tx = x-1;

        }

        else{

            x -= 1;

            y += 1;

            d1 = d1 - t4bb \* x + t4aa \* y + t2aa;

            tx = x;

        }

    }

    var d2 = t2bb \* (x\*x + 1) - t4bb \* x + t2aa \* (y \* y + y - tbb) + taa / 2;

    while(x>=0){

        CirclePoint4(x,y,x0,y0,color);

        if(d2<0){

            x-=1;

            y+=1;

            d2 = d2 + t4aa \* y - t4bb \* x + t2bb;

        }

        else{

            x-=1;

            d2 = d2 - t4bb \* x + t2bb;

        }

    }

}

第四章-多边形填充

1 多边形的扫描转换

多边形的扫描转换就是对多边形内部进行填充，主要用于绘制物体的表面模型。

1.1多边形的定义

多边形是由折线段组成的封闭图形，可以分为凹多边形，凸多边形，环。

1. 凸多边形

多边形上任意两点间的连线都在多边形内，凸点对应内角和小于180。

1. 凹多边形

多边形上存在有两顶点间的连线不在多边形内部，凹点对应内角和大于180.

1. 环

多边形内包含有另外的多边形。

1.2 多边形的表示

1. 顶点表示法 用多边形的顶点序列来描述，特点是直观，占内存少，易于进行几何变换，需要对多边形进行扫描转换。
2. 点阵表示法 用多边形覆盖的像素点集来描述。是填充多边形所需要的表示形式。
3. 多边形的扫描转换 将多边形的描述从顶点表示法变换到点阵表示法的过程，称为多边形的扫描转换。

1.3 多边形的着色模式

多边形可以使用平面着色模式，光滑着色模式。

1.4 多边形填充算法

从多边形的顶点信息出发，求出其覆盖的每个像素点，取为填充色，将多边形外部保留位背景色。主要工作是确定穿越多边形扫描线的覆盖区间，然后将其着色。首先确定多边形覆盖的扫描线条数，对每一条扫描线，计算其与多边形边界的交点区间，然后将区间内像素着色。

1.5 区域填充算法

区域是指一组相邻而又具有相同属性的像素，可以理解为图形的内部。

区域一般由封闭边界定义。区域的边界色和填充色不一致，区域一般采用种子填充算法。

种子填充算法是从区域内给定的种子位置开始，按填充颜色填充种子的相邻像素直到边界，主要有4邻接点算法，和8邻接点算法。

2 有效边表填充算法

2.1 填充原理

按照扫描线从小到大的移动顺序，计算当前扫描线与多边形各边的交点，然后把这些交点按x值递增排序配对，确定填充区间，然后用指定颜色填充像素。

2.2 边界像素的处理原则

需要考虑到像素面积大小的影响，若边界上的像素全部填充，则像素覆盖面积大于实际面积。

为了解决这些问题，采用“下闭上开”，“左闭右开”的原则处理边界像素。

总结：填充左下边界的像素，忽略右上边界的像素

上图顶点可分为三类：

1. 局部最高点 P1, P6, P4，共享顶点的两条边在扫描线下方，顶点取为0
2. 普通连接点 P2，共享顶点的两条边在扫描线上下两侧，顶点取为1
3. 局部最低点 P0, P3, P5 共享的两条边在扫描线上方，顶点取为2
4. 普通连接点处理原则

P2作为 P3 P2 边的终点，同时作为 P2 P1 的起点。按照“下闭上开”，P2 作为 P3 P2 边的终点不填充，但作为P2 P1 的起点被填充

1. 局部最低点处理原则

共享 P3 的 P3 P2 边和 P3 P4 边加入有效边表。

1. 局部最高点处理原则

顶点个数计数为0，扫描线自动填充 P4, 根据“下闭上开” 自动放弃局部最高点，P1 P6不被填充， P4被y=5 填充

2.3 有效边和有效边表

2.3.1 有效边

多边形内与当前扫描线相交的边成为有效边（AE）。有效边交点之间有相关性，容易用增量法计算出下一个扫描线于有效边的交点。

2.3.2 有效边表

将有效边按照于扫描线交点的x坐标递增的顺序存放在一个链表中，成为有效边表（AET）

3 边缘填充算法

3.1 填充原理

求出多边形每条边于扫描线的交点，然后将交点右侧所有像素颜色取反。处理完所有边后就完成了多边形填充任务。

4 区域填充算法

4.1 填充原理

对于用点阵表示的区域，如果其内部像素具有同一种颜色，而边界像素具有另一种颜色，可以用种子算法填充。种子算法是在区域内任何一个种子像素开始，由内向外将颜色扩散到整个区域。

4.2 四邻接点和八邻接点

1. 四邻接点定义

对于区域内部任何一个种子像素，其左，上，右，下这四个像素称为四邻接点

2. 八邻接点定义

对于区域内部任何一个种子像素，其左，上，右，下，左上，右上，右下，左下，这八个像素称为八邻接点

多边形的扫描转换

\* 多边形有两种重要的表示方法：顶点表示和点阵表示。

\* 顶点表示是用多边形的顶点序列来表示多变形。这种表示直观、几何意义强、占内存少，易于进行几何变换，被广泛用于几何造型系统中。

\* 点阵表示是用于多边形内的像素几何来刻画多边形。这种表示丢失了许多几何信息（如边界、顶点），但它是光栅显示图形所需要的表示形式。

\* 多边形的扫描转换：把多边形的顶点表示转换为点阵表示

\*\*使用的算法有：\*\*

\* 多边形的扫描转换

\* 扫描线算法

    \* 边缘填充算法

    \* 边界标志法

\* 区域填充算法

1. 多边形的扫描转换算法

1.1 扫描线算法

目标：利用相邻像素之间的连贯性，提高算法效率。

处理对象：非自交多边形（边与边之间除了顶点外无其他交点）。

基本思想：按扫描线顺序，计算扫描线与多边形的相交区间，再用要求的颜色显示这些区间的像素，即完成填充工作。

四个步骤：

1. 求交：计算扫描线与多边形各边的交点

2. 排序：把所有交点按x值递增顺序排序

3. 配对：第一个与第二个，第三个与第四个等待；每对交点代表扫描线与多边形的一个相交区间

4. 着色：把相交区间内的像素置成多边形颜色，把相交区间外的像素置成背景色

\*\*存在问题1\*\*：当扫描线与多边形顶点相交时，交点的取舍问题。

\_解决方法：\_

当扫描线与多边形的顶点相交时

\* 若共享顶点的两条边分别落在扫描线的两侧，交点只算\*\*一个\*\*

\* 若共享顶点的两条边在扫描线的同一侧，这时交点作为\*\*零个或两个\*\*

\* 具体实现时，只需检查顶点的两条边的另外两个端点的y值。按这两个y值中大于交点y值的个数是0，1，2来决定。

\*\*存在问题2\*\*：边界上像素的取舍问题，避免填充扩大化。

\_\_解决方法\_\_：边界像素：规定落在右上方边界的像素不予填充。具体实现时，只要对扫描线与多边形的相交区间\*\*左闭右开\*\*。

\*\*边相关扫描线填充算法\*\*

扫描线的相关性质：某条扫描线上相邻的像素，几乎都具有同样的内外性质，这种性质只有遇到多边形边线与该扫描线的交点时才会发生改变。

边的相关性：由于相邻扫描线上的交点是与多边形的边线相关的。对同一条边，前一条扫描线yi与该边的交点为xi，而后一条扫描线yi+1 = yi + 1，与该边的交点则为xi+1 = xi + 1/m，利用这种相关性可以省去大量的求交运算。

边相关扫描线算法的实现需要建立两个表：\*\*新编表（NET）\*\*和\*\*活动边表（AET）\*\*。

新编表的构造：

1. 首先构造一个纵向扫描线链表，链表的长度为多边形所占有的最大扫描线数，链表的每个结点称为一个桶，对应多边形覆盖的每一条扫描线。

2. 将每条边的信息链入与该边最小y坐标（ymin）相对应的桶处。也就是说，若某边的较低端点为ymin，则该边就在相应的扫描线桶中。

3. 每条边的数据形成一个结点，内容包括：该扫描线与该边的初始交点x（即较低端点的x值），1/k，以及该边的最大y值ymax。

4. 同一个桶中若干条边按ymax由小到大排序，若ymax相等，则按照1/k由小到大排序。

活动边表的构造：

NET表建立以后，就可以开始扫描转换了。对不同的扫描线，与之相交的边线也是不同的，当对某一条扫描线进行扫描转换时，我们只需要考虑与它相交的那些边线，为此需要建立一个只与当前扫描线相交的边记录链表，称之为活动边表。

合并NET和AET：

\*\*算法过程\*\*：

1. 根据给出的多变形顶点坐标，建立NET表，求出顶点坐标中最大y值ymax和最小y值ymin。

2. 初始化AET表指针，使它为空。

3. 执行下列步骤直至NET和AET都为空：

> 3.1 如果NET中的第y类非空，则将其中的所有边取出并插入AET中；

> 3.2 如果有新边插入AET，则对AET中各边排序；

> 3.3 对AET中的边两两配对，（1和2为一对，3和4为一对，……），将每对边中x坐标按规则取整，获得有效的填充区段，在填充；

> 3.4 将当前扫描线纵坐标y值递增1；

> 3.5 如果AET表中某记录的ymax=yi，则删除该记录（因为每条边被看作下闭上开的）；

> 3.6 对AET中剩下的每一条边的x递增1/k，即x+=1/k

第五章-图形几何变换

1. 二维图像剪裁

1.1 图形学中常用的坐标系

1. 世界坐标系：描述现实世界中场景的固定坐标系统。
2. 用户坐标系：描述物体几何模型的坐标系，主要用于建立物体的几何模型。
3. 观察坐标系：世界坐标系中定义的直角坐标系，二维观察坐标系用于指定图形的输出范围。
4. 屏幕坐标系：实数域二维或三维坐标系。
5. 设备坐标系：光栅扫描显示器等图形设备自带一个二维直角坐标系。
6. 规格化坐标系

2. Cohen-Sutherland直线段剪裁算法

在二维观察中，需要在观察坐标系下根据窗口大小对世界坐标系中的二维图形进行裁剪，只将位于窗口内的图形变换到视区输出。

基本思想

首先判断直线段是否全部在窗口内，是，则保留；不是，则再判断是否完全在窗口之外，如是，则舍弃。如果这两种情况都不属于，则将此直线段从交点处分割，对分割后的线段再进行如前判断。直至所有直线段和由直线段分割出来的子线段都已经确定了是保留还是舍弃为止。

编码

对直线进行区域编码，将裁剪面所在的平面划分成九个区域

用四个二进制位对九个区域编码，C3 C2 C1 C0

1. C0：该端点位于窗口左边时取1
2. C1：该端点位于窗口右边时取1
3. C2: 该端点位于窗口下边时取1
4. C2: 该端点位于窗口上边时取1

裁剪步骤

* 若直线两端区域编码都为0，(C1 | C2 = 0) 说明线段在窗口内，简取
* 若直线两端点区域编码都不为0，(C1 & C2 != 0) 说明两端点都在裁剪面外，舍去
* 其他情况：
  + 将线段与裁剪面的四个边界进行求交，得到交点作为新的顶点继续判断是否在窗口内部或者全部在窗口外部。

1.2 边缘填充算法

1.3 边界标志算法

2. 区域填充算法

区域：指已经表示成点阵形式的填充图形，它是象素的集合。

表示方法：内点表示、边界表示

内点表示

\* 枚举处区域内部的所有像素

\* 内部的所有像素着同一个颜色

\* 边界像素着与内部像素不同的颜色

边界表示

\* 枚举出边界上所有的像素

\* 边界上的所有像素着同一颜色

\* 内部像素着与边界像素不同的颜色

区域填充 —— 对区域重新着色的过程

\* 将指定的颜色从种子点扩展到整个区域的过程

\* 区域填充算法要求区域是连通的

连通性：4连通、8连通

二维直线的裁剪

直线的裁剪有如下算法：

1. Cohen-Sutherland 算法

2. 重点算法

3. Liang-Barsky 算法

多边形的裁剪有如下算法：

1. Sutlerland-Hodgman 算法

2. Weiler-Athenton 算法

Cohen-Sutherland 算法

思想：对于每条线段P1P2分为3种情况处理：

1. 若P1P2完全在窗口内，则显示该线段P1P2，简称“取”之

2. 若P1P2明显在窗口外，则丢弃该线段，简称“弃”之

3. 若线段既不满足“取”的条件，也不满足“弃”的条件，则在交点处把线段分为两段，其中一段完全在窗口外，可弃之，然后对另一段重复上述处理。

为了方便计算机判断窗口与直线段属何种关系，采用如下编码方式：

1. 延长窗口的边，将二维平面分成9个区域

2. 每个区域赋予\*\*4位\*\*编码